

LINUX EXPERT

FAQ

Q I've noticed that when I want to install a program and download its RPM installer file, there's sometimes an md5 checksum printed on the download page. What is it for?

A A checksum is a unique code that has been generated using something called the message-digest-5 algorithm. If you generate a checksum for a file, it is guaranteed to be unique. If you change the original file slightly, even by one bit of one byte, and then generate another checksum it won't match the first one.

There is a risk that servers hosting downloadable software may be hacked, and the attacker could replace packages with those that contain backdoors. If you installed one of these altered programs, the attacker may be able to gain control of your computer. The md5 checksum associated with the package serves as a guarantee that the package on the server is what it says it is. To check the md5 checksum, enter the following command:

```
# md5sum <downloaded package>#
```

The resulting alphanumeric string is the checksum, so for instance:

```
563c1bfff307a16d45f5d6a04011f07b
```

Jon Thompson
UNIX/Linux management and security consultant

linux@computershopper.co.uk

Customising your bootup

Seize control of your Linux environment with the help of the startup system. Jon Thompson gets environmentally friendly by setting up different boot sequences in Linux

The Linux startup system is flexible and provides lots of ways in which you can customise your computer. For a start, you can use it to create a set of working environments, each running different groups of programs. It is possible to switch between these environments at will and without rebooting.

Maybe every morning you want to start your web browser and go to a news website and run your email program to sift through your messages while loading up OpenOffice. At lunchtime, a single command can close those programs and open a chat client instead.

In this month's *Linux Expert*, we look at how to achieve this level of customisation by delving into the mysterious world of initab. We also show you how to make your servers wake up and boot by contacting them with a modem or sending a signal over the local network.

BOOT CAMP

When a PC boots, its BIOS searches for a boot device. Once found, it loads and runs the first stage of the boot loader from the device's master boot record. The two most common boot loader programs are Grub and Lilo. Being only 446 bytes long, this first stage is usually used to load and run the second, much larger stage. This is found in the first active partition on the disk and, in turn, presents a set of boot options.

The boot loader then fetches the selected kernel, which is stored as a compressed image a bit like a Zip file. It runs a small piece of code at the head of the image to inflate (or unzip) the kernel into RAM and then gives the kernel control of the computer. This begins the Linux boot process. Now it has control, the kernel initialises any co-processors present on the computer, initialises the RAM and scans the hardware

for storage devices and other peripherals. It then loads the required drivers to run the hardware, loads various required kernel modules and finally mounts the file systems on the hard disks.

All these activities happen quickly and their output tends to scroll up the PC's screen very fast. Many Linux distributions hide the kernel's output behind a splash screen, although pressing the Esc key should display the lines of text. Additionally, the kernel always makes a copy of its output for troubleshooting purposes. You can see it by running the following command:

```
# dmesg | more#
```

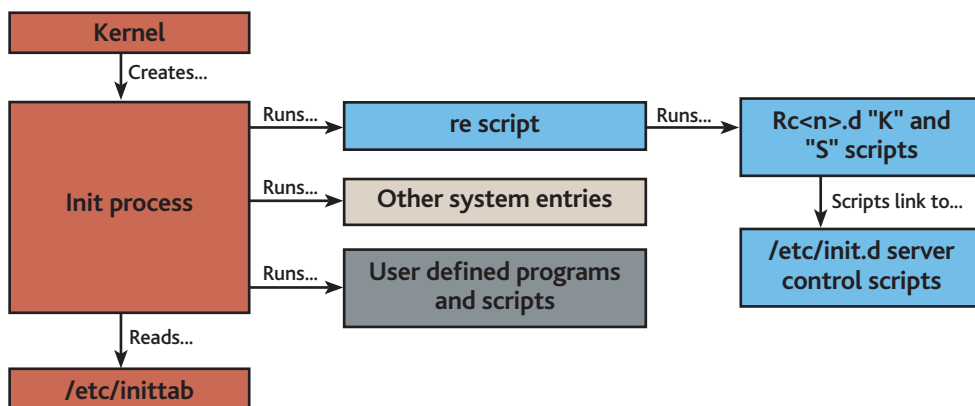
This long list shows that the kernel works hard during bootup. The dmesg command is useful if you're adding an unusual peripheral, such as a USB Freeview tuner as we did in *Linux Expert*, *Shopper* January 2007. You can see how the kernel treats the new device, whether it worked, or what error messages appear.

INTRODUCING INIT

When it's done, the kernel starts its scheduler, which is the software that keeps track of the running processes on the system and allocates each one with time on the processor. It then creates just one process, called init. Init has the responsibility of starting Linux's services, logging process, daemons to handle terminal inputs and any user-specified software.

The init process won't respond to a 'kill -9' command, which should stop dead any running process on the system in its tracks. This is because init continues to play an active role right up until shutdown. For example, it adopts processes whose parents have died. This situation happens when a process (the

Starting up The Linux boot process



▲ When it boots up, a Linux system runs through a number of processes. The init process is at the core of the startup procedure and initialises other processes. These include critical scripts that get the computer up and running, as well as less important things such as the programs that users want to have start automatically for their convenience



parent) starts another process (the child) and then terminates, leaving the new process running on the system. Init also helps reconfigure the system to run in several different ways, called runlevels. We'll look at runlevels in depth shortly, but for now think of them as being the different modes in which Linux can run.

Even for seasoned Linux experts, the content of the `/etc/inittab` file, used to control the init process, remains a mystery. But it isn't that hard to understand because it has a simple structure and a manageable set of options. The init process works through the directives stored in the inittab file, running scripts and starting software that pertains to the current runlevel.

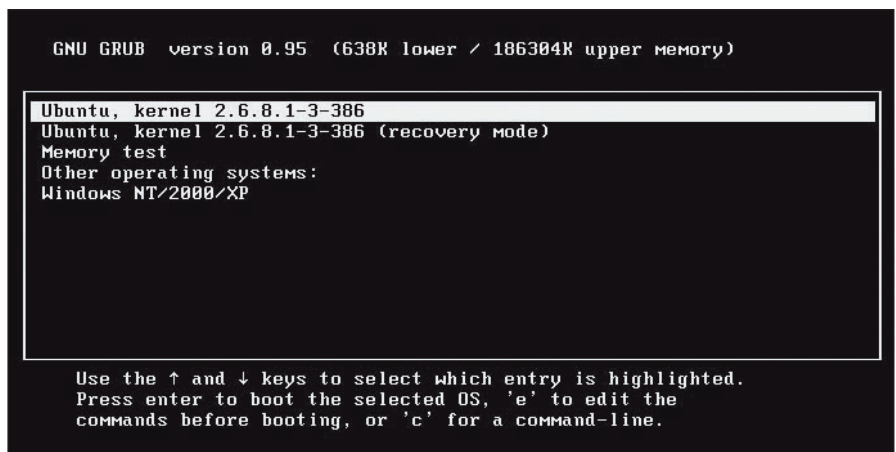
You can tell it to start programs in a number of ways, perhaps running some only once and at boot time, for instance, while others need to be respawned if they die unexpectedly or have to be re-run after use.

ON THE LEVEL

Central to inittab, and to Linux itself, is the concept of runlevels. Understanding these is central to understanding how inittab works. There are seven runlevels generally available to a Linux system, numbered 0 to 6. See the 'Level best' box below for a short description of those available. Most people care about only five of these because two (0 and 6) relate to the state of the system when it is shut down and when it is rebooting.

Each runlevel has its own specific use and the different runlevels indicate to init what services and servers the machine should have running. Runlevel 1, for example, is mostly used on larger servers when performing maintenance duties. It gives you a text-only console and allows network access so you can do such things as mount remote file systems over NFS. However, it prevents remote users logging in. This makes it perfect for those times when you need to carry out software upgrades or emergency backups on a live network. Runlevel 2 is also a single-user, text-only mode, but it doesn't allow any networking support.

Servers that sit in a data centre and serve as storage areas for files don't usually need full graphical desktop software, so runlevel 3 is used. This allows full, multi-user network access but text-only console access. Runlevel 4 is not officially used, though there's no reason why you can't use it yourself. Most people, however, only ever boot into runlevel 5. On a commercial or domestic Linux distribution, this provides the usual graphical desktop and networking support.



▲ Booting straight into a particular runlevel is an involved process with Grub, but if you confirm you want to leave graphical mode you should see the boot options listed in text form

If you were to enter runlevel 0, the system would shut itself down immediately, and usually turn off the power too, whereas runlevel 6 will cause the computer to reboot itself immediately and come back up. Runlevel 6 is synonymous with the reboot command, while 0 is the same as entering a shutdown such as the following:

```
# shutdown -h now
```

The `-h` switch tells the shutdown command to halt the processor once it has completed. The 'now' parameter means that shutdown should happen straight away. If you want to delay shutdown, you can provide a time in seconds instead of 'now'. For practical reasons, you should never set the default runlevel to 6, and some distributions will check and change it to 5 instead so the system doesn't simply reboot itself repeatedly.

On a running system, the root user can use the init command to switch between the different runlevels:

```
# init 6
```

This command reboots the system, whereas the following puts it into single-user, text-only mode. Always be sure to save any work in any running graphical desktop applications before you enter any of these commands:

```
# init 1
```

The inittab file contains a line defining a default runlevel, usually placed somewhere near the top

of the file. This default is usually runlevel 5 unless your Linux computer is a server. You can, however, override this default by booting into a different runlevel using the facilities provided by a bootloader such as Grub and Lilo. We'll explain how to enter runlevel 1.

Booting straight into a particular runlevel depends on the boot loader you have installed on your system. Lilo is the easiest. At the 'boot:' prompt, simply enter the command:

```
boot: linux 1
```

This will boot Linux into runlevel 1, which is the single-user, text-only mode with networking. This change is temporary, and the default runlevel will come back into effect when you next boot the system.

This process is a little more involved with Grub and may depend slightly on your distribution. Here's how it works on a Novell SUSE 10 system. Press Esc when presented with the usual graphical boot options. After confirming that you want to leave graphical mode, you'll see the same boot options presented in text form (shown above). These may include Linux, Windows and other operating systems if you have them installed. Scroll down to the desired boot option ('Linux' or similar) and press 'e' to edit it.

Some further options will appear. Scroll to the kernel line and press 'e' to edit it. At the end of the line, add a space and 1, then press Return. Press Esc to go back to the choice of operating system, select the one you've just edited, and press Return again to boot the computer. As with Lilo, changing the runlevel in Grub this way is just temporary, so the next reboot will start the system using its default runlevel.

Level best Runlevels explained

A Linux system can boot into one of several modes, called runlevels. There are seven of these, including shutdown and reboot, each used for a specific task. Here's a list of the available runlevels, with a short description of their capabilities or restrictions:

Runlevel	Description
0	System halted. Will usually turn off the power, too
1	Single-user text-only mode with networking support
2	Single-user mode without networking support
3	Multi-user text-only system with networking support
4	Not used
5	Full graphical multi-user mode with networking support
6	Reboot the system

Most desktop Linux systems boot into runlevel 5.

START YOUR SERVERS

When you run the init command on a working system to change runlevels, the init process consults inittab to see which services should be running when that runlevel is reached. If you look at the contents of inittab, you'll see a series of entries such as the following:

```
10:0:wait:/etc/init.d/rc 0
11:1:wait:/etc/init.d/rc 1
12:2:wait:/etc/init.d/rc 2
13:3:wait:/etc/init.d/rc 3
14:4:wait:/etc/init.d/rc 4
15:5:wait:/etc/init.d/rc 5
16:6:wait:/etc/init.d/rc 6
```

We'll come to what the different fields mean in a moment. Notice that the second line relates to runlevel 1, which we've tried to reach by editing the boot loaders. These lines say when entering a desired runlevel, you run a command and wait for it to finish. In this case, the command is the `/etc/init.d/rc` script, with a parameter indicating the new runlevel.

The `rc` script uses an arrangement of directories and control scripts that ensure the services you expect to find running at each runlevel start up properly, and those you don't want won't be started. There's no point starting the Apache web server, for example, if you're working in runlevel 2, which doesn't provide network support.

You can use `inittab` to make `init` start your own programs and daemons in several useful ways. Let's take a closer look at a typical entry:

```
ab:5:respawn:mydaemon
```

The directive splits into four fields, separated by colons. The first directive is just a unique, two-letter identifier. This can be anything as long as it's unique in the `inittab` file. The next field is a list of the runlevels at which the program should run. You specify the runlevels as a string. For example, 135 means that the program should run when the system starts in runlevels 1, 3 or 5. `Respawn` is the action `init` should take when handling the process, and finally we have the command itself, including any arguments.

The official names of the fields are:

```
<id>:<runlevels>:<action>:<process>
```

The action field gives us a good degree of freedom to tune exactly how to run and re-run the program. `Respawn`, as we've used above, tells `init` to re-run the program if it dies, but how does `init` know when this happens?

`Init` is the ancestor of all processes. Those started in response to lines in `inittab` are its

children and so, when they die, it immediately receives a signal from the kernel telling it so. If that happens, it will re-run the terminated program. There are a number of good reasons for using `respawn`, not least of which is the ability to resurrect software that occasionally crashes.

In `respawn`'s place, the `'wait'` action would make `init` wait until the program in question has finished, which is useful if you have programs that need to run in a particular order and finish before moving on. You might, for instance, want to check the integrity of a database, and correct it if necessary using a housekeeping program, before running the database server itself.

The `'once'` action instructs `init` to run the program once only when entering any of the specified runlevels, whereas the `'boot'` action does the same but only at boot time. The `'bootwait'` action ensures that `init` runs the program only at system boot time, then waits for it to finish before moving on. In the cases of boot actions, the runlevels are ignored.

`Telinit`, used in place of the `init` command, enables you to tell the `init` process to run `inittab` entries with runlevels of a, b and c:

```
# telinit a
```

When you use `telinit`, the computer stays at the current numeric runlevel (runlevel 5, say).

`Telinit` is useful in cases where you want to run a number of programs together but only at certain times, without manually opening and closing them. It enables you to set up three different user environments so you might include an entry in `inittab` that runs `OpenOffice` and `Evolution` when the system is set to runlevel a. Running `telinit` with a parameter of 'a' would set you up for the working day. Runlevel b might start your favourite games and a media player.

If you've used `telinit` to run programs associated with runlevel a and you subsequently use it to run those associated with runlevel b, the programs specified for runlevel a will be shut

down before those associated with runlevel b start up. Continuing with the possible action field values, lines with an action of `'sysinit'` will execute at system boot time, but before the boot or `bootwait` entries.

There are also a few power-aware actions you can use when things go wrong. The `'powerwait'` action tells `init` to execute the associated program or script immediately when the power fails. That sounds nonsensical, but it's for use on systems with an uninterruptible power supply (UPS) that takes over during a blackout. The `'powerwait'` action is most useful in cases where you need to invoke a script to shut down a database server or similar gracefully, so that it doesn't lose any buffered data. `Init` will wait for the process to finish before processing other entries. Similarly, the `'powerfail'` action specifies a script or program that runs at a power failure, though this time `init` doesn't wait for it to finish.

If you have a UPS and power is restored following a power cut, the `'powerokwait'` action will run the specified process when `init` receives a signal to say that the mains is back on. This might involve restarting the database server. If the power doesn't come back on and the UPS batteries start to fail, then entries with a `'powerfailnow'` action will execute, giving you a last chance to tell your servers to exit gracefully and the system to shut down.

If you have a notebook, you can make specific programs run when the system resumes after time spent in sleep or suspend mode using the resume action. Similarly, on any computer, you can have programs run in response to a `Ctrl-Alt-Del` sequence with the `ctrlaltdel` action. This could be a warning message or a proper shutdown command, rather than the potentially data-damaging situation that occurs when the user kills the system manually. See the 'Boot, init, action' box (left) for an overview of the available actions.

SERVER, RUN THYSELF

When you boot or change into certain runlevels, some servers start, while others stop. But how does the `rc` script know what to do? The answer lies in the mysterious and often-ignored `rc.d` directories.

Each runlevel has an associated `rc` directory that exists within the `/etc/init.d` directory. These sub-directories are called `rc0.d` through to `rc6.d`. On some distributions, such as `Ubuntu` and others based on `Debian`, they are stored in the `/etc` directory.

Take a look at one of the `rc` directories with the command `'ls -l'`, and you'll see that it is populated with links to scripts in the `/etc/init.d` directory. These are the control scripts for the various installed servers. Some of the links have the prefix `S` for start, and others have the prefix `K` for kill. They also have a number, indicating the order in which they should be run when entering and exiting that runlevel.

It is possible to add our own lines directly to `inittab`. For instance, to start the `apache2` web server for use in runlevels 3 and 6, without respawning if it fails, you could use the line:

```
ap:36:once:/etc/init.d/apache2 start
```

On some distributions, notably those that are not based on `Debian`, there's an easier way of going about things. If your distribution comes with the `inserv` package, install it and then

Boot, init, action Controlling the system

The actions field in `inittab` can take the following values, giving you fine control over how the system works in response to boot and power events:

Action	Description
Respawn	Restart the process if it dies
Wait	Pause processing the <code>inittab</code> file until the process completes
Once	Execute once upon entering the specified runlevels
Boot	Only execute at system boot (runlevels are ignored)
Bootwait	Execute at boot and wait for completion before going on
Off	Does nothing
Ondemand	Used with <code>telinit</code>
Initdefault	Defines the default runlevel
Sysinit	Runs at boot time, before boot and <code>bootwait</code> actions
Powerwait	Execute first when the power fails (requires a UPS)
Powerfail	Execute when the power fails (requires a UPS)
Powerokwait	Run first when power is restored
Powerfailnow	Executed when the UPS's batteries start to fail
Resume	Executed when resuming after being suspended (used mainly on notebooks)
Ctrlaltdel	Execute when the user presses the <code>Ctrl-Alt-Del</code> key combination

The most common action is `'once'` and it is used when you want a program to run just once, thereby letting it die if it fails. The `'respawn'` command is also popular, running a program and then re-running it again every time it dies.

The power-management options require that you have an uninterruptible power supply (UPS) that has a Linux driver, so it can indicate to the `init` process that mains power has failed and it has switched to battery backup, or that the power has been restored.



install the Apache2 package from your distribution media. Now enter the following to see the header in its control script:

```
# more /etc/init.d/apache2

### BEGIN INIT INFO
# provides: apache2 httpd2
# Required-Start $local $remote_fs
# $network
# X-UnitedLinux-Should-Start: $named
# $time postgresql sendmail mysql
# ypcclient dhcp radiuds
# Required-Stop: $local_fs $remote_
# fs $network
# X-UnitedLinux-Should-Stop:
# Default-Start: 3 5
# Default-Stop: 0 1 2 6
# Description Start the httpd daemon
# Apache 2
### END INIT INFO
```

The important lines are those containing the keywords Default-Start and Default-Stop. It's the job of the insserv command to read this header information and create K and S links to the apache2 script in the relevant rc directories.

This header contains a number of variables, such as \$local and \$named. Their definitions come from /etc/insserv.conf, which insserv reads whenever you run it. To run insserv, and thereby install apache so it runs at boot time, enter the following command:

```
# insserv /etc/init.d/apache2
```

Insserv creates links beginning with S in each of the runlevel directories indicated in Default-Start, and creates those beginning with K in the runlevel directories indicated in Default-Stop. When you next change to a relevant runlevel, the rc script executes the links beginning with K in the rc directory for the runlevel you're leaving, and then runs those beginning with S in the rc directory you're entering.

The rc script does this by compiling a sorted list. This is because there's a two-digit number in the name of each link. This provides a method of deciding the order in which the various servers and services should shut down and start up.

You can also use insserv to uninstall a server at any time. Enter the following command:

```
# insserv -r /etc/init.d/<server
# control script>
```

This will cause insserv to re-read the header information in the indicated control script and delete the affected S and K links from the relevant runlevel directories.

As an efficiency measure, in cases when you change runlevels and a particular server is available in both, the rc script knows enough to realise that this server is not to be shut down and restarted. Once rc has changed runlevels, it continues reading inittab directives, looking for any lines that specify programs that should run in the target runlevel.

REMOTELY YOURS

Look at the back of practically any desktop computer, and many well-specified notebooks, too, and you'll find a couple of D-shaped, nine-pin serial ports that you probably never use. If you have a terminal, or a computer running a

terminal program, you can log into this serial-equipped computer remotely over these ports.

There's good reason for wanting to do this, even though it provides only text-based access. If you configure your firewall to prevent all remote access, you can still control the system over this serial link. All you have to do is press return on whatever's connected to the serial line and you'll get a login prompt. Here's how to configure it.

First, run the following command to make sure your system recognises the serial ports:

```
# dmesg | more tty
ttyS0 at 0x03f8 (irq = 4) is a 16550A
ttyS1 at 0x02f8 (irq = 3) is a 16550A
```

In this example, the two ports are recognised by the kernel as ttyS0 and ttyS1. Now you need to add the following lines to inittab:

```
s0:12345:respawn:/sbin/agetty -f
/etc/issue serial 9600 ttyS0 vt100
s1:12345:respawn:/sbin/agetty -f
/etc/issue serial 9600 ttyS1 vt100
```

These lines tell init to run the agetty program and have it listen on both serial ports at runlevels 1 through to 5. No matter what runlevel you boot into, you'll still have physical access.

The argument '-f /etc/issue serial' lets you provide a customised message to the person logging in. It tells agetty to send the contents of the /etc/issue serial text file when someone presses return to log in. You can make this display your company name, a dire warning against abuse or whatever you like.

The numbers following this argument are the baud rate. The higher the number, the faster the connection. Though many remote terminals will automatically detect this rate, you must tell agetty which to use. You can choose from the following values: 110, 300, 1200, 2400, 4800, 9600, 19200, 28400, 57600, 115200. As you only intend to enter and receive text, you don't really need this setting to be the highest, so 9600 will be fine.

Finally, there's the terminal type. This refers to the control characters to which the terminal responds. These can be used to do things such as clearing the screen. Our example setting, vt102, refers to the industry standard DEC VT102 terminal type. Most terminal emulation packages support this by default.

If you have a modem attached to one of your serial ports, you can call it and log in to the serial port. This has some interesting remote management uses and, if you have a correctly configured motherboard, you can even boot the machine remotely and log in.

WAKE UP

As well as customising the inittab file, it's also possible to control a properly configured computer to boot by simply contacting it over a modem or over the LAN. In situations where the server is difficult to get at, or is used rarely, this can save lots of time. This wake-on-LAN (WOL) and wake-on-modem (WOM) capability isn't present on all systems and requires hardware with certain capabilities. When it is present, it can prevent you needing to visit a server personally just to switch it on and boot it up.

First, your BIOS and motherboard must have WOM or WOL capability, and these options must be enabled in the BIOS. You can check for

compatibility and enable these options by checking the power-management settings of your BIOS when you boot the computer. A good indication that the BIOS will have these capabilities is that the motherboard will be PCI 2.1-compliant.

Your power supply must also be ATX-compliant and your network interface card must be WOL-compatible to use the wake-on-LAN feature. If it is, you may have been supplied with a WOL cable you should connect between the network card and the motherboard. Some network cards don't need such a cable, however. See your hardware user guide for more details. You'll also need to make a note of the MAC address of the network card before shutting down the server. You can obtain this using the ifconfig command when logged in as root:

```
# ifconfig eth0
```

The MAC address is the 'Hwaddr' entry near the top of the listing and consists of six hexadecimal numbers separated by colons.

The way WOL works is that power continues to be supplied to the network card after you shut down and switch off the system. One or more of its LEDs should still be lit or blinking, as should the LED on the hub or router corresponding to the port it's using. This indicates that the ATX-compliant power supply is still keeping the card alive.

With the server shut down, log on to another Linux computer and install the network diagnostics (netdiag) package from your distribution media. Enter the following command, where <MAC address> is the MAC address of the network card you took a note of just now:

```
# ether-wake <MAC address>
```

This sends a so-called 'magic packet' to the server, consisting predominantly of its network card's MAC address. These addresses are globally unique and built into the hardware, so there's no danger of starting the wrong server by mistake.

The server's cooling fans should start to blow, the disks will spin up, the BIOS should issue a beep and the boot sequence will spring into life as the computer runs through its usual bootup sequence. Similarly, in the case of a wake-on-modem-compatible system, the motherboard never truly switches off but continues to take power and waits for a signal on the modem line telling it to boot.

So, rather than having to start pieces of software by hand every time you boot your servers, a combination of inittab, the init process and the rc script will enable you to do so automatically.

When the power fails, you can use inittab to close down the system gracefully. However, when the power returns, the hardware may not automatically reboot. This is where wake-on-LAN or modem features come into their own. For many Linux administrators, they can mean the difference between getting back to sleep quickly, and long, sleepy car journeys to press a simple power-on button. ☐

Next month

ESSENTIAL LINUX COMMANDS

Discover the hidden power of your system